The Ruthless Pursuit of the Truth about COTS

Dr. Norman F. Schneidewind
Naval Postgraduate School
2822 Racoon Trail
Pebble Beach
California, 93953, USA
Email: nschneid@nps.navy.mil

Abstract

We expose some of the truths about COTS, discounting some exaggerated claims about the applicability of COTS, particularly with regard to using COTS in safety critical systems. Although we agree that COTS has great potential for reduced development and maintenance time and cost, we feel that the advocates of COTS have not adequately addressed some critical issues concerning reliability, maintainability, availability, requirements risk analysis, and cost. Thus we illuminate these issues, suggesting solutions in cases where solutions are feasible and leaving some questions unanswered because it appears that the questions cannot be answered due to the inherent limitations of COTS. These limitations are present because there is inadequate visibility and documentation of COTS components.

Introduction

In this paper we analyze three important aspects of COTS software: 1) reliability, maintainability, and availability; 2) requirements risk assessment, using risk factors from the Space Shuttle and modifying them for more general use; and 3) cost framework. We are motivated to address these issues because we feel that the COTS community has not adequately addressed some very important questions concerning the applicability of COTS when used in a host system. We define a host system as follows: it contains both COTS and non-COTS software; the latter is specific to the operational mission of the organization; and the mission cannot be satisfied entirely by COTS components. Our concerns are reinforced by Kohl: "The most significant challenges of V&V of COTS products has to do with knowledge of the functionality, performance and quality of these products. Because these products tend to be developed for large, commercial markets as opposed to being developed to a specification for a single customer, they tend to provide a variety of useful and desirable features for the market that they are targeted for, at the expense of the specific system needs in which such products may be used. Further, quality and reliability are sometimes not considered critical when time-to-market is a driving requirement. Thus, it is sometimes the case that these COTS products contain features and functionality that may not be fully known, even to the vendor." [KOH99].

Many vendors produce products that are not domain specific (e.g., network server) or have limited functionality (e.g., mobile phone). In contrast, many customers of COTS develop systems that are domain specific (e.g., target tracking system) and have great variability in functionality (e.g., corporate information system). This discussion takes the viewpoint of how the customer can ensure the quality of COTS components. In addition to direct quality evaluation, we also consider requirements risk analysis in a later section, which indirectly affects quality. We must distinguish between using a non-mission critical application like a spreadsheet program to produce a budget and a mission critical application like military strategic and tactical operations. Whereas customers will tolerate an occasional bug in the former, zero tolerance is the rule in the latter. We emphasize the latter because this is the arena where there are major unresolved problems in the application of COTS. Furthermore, COTS components may be embedded in host systems. These components must be reliable, maintainable, and available, and must interoperate with the host system in order for the customer to benefit from the advertised advantages of lower development and maintenance costs. Interestingly, when the claims of COTS advantages are closely examined, one

Paper presented at the RTO IST Symposium on "Commercial Off-the-Shelf Products in Defence Applications "The Ruthless Pursuit of COTS"", held in Brussels, Belgium, 3-5 April 2000, and published in RTO MP-48.

finds that to a great extent these COTS components consist of hardware and office products, not mission critical software [CLE97].

Obviously, COTS components are different from host components with respect to one or more of the following attributes: source, development paradigm, safety, reliability, maintainability, availability, security, and other attributes. However, the important question is whether they should be treated differently when deciding to deploy them for operational use; we suggest the answer is *no*. We use *reliability* as an example to justify our answer. In order to demonstrate its reliability, a COTS component must pass the same reliability evaluations as the host components, otherwise the COTS components will be the weakest link in the chain of components and will be the determinant of software system reliability. The challenge is that there will be less information available for evaluating COTS components than for host components but this does not mean we should despair and do nothing. Actually, there is a lot we can do even in the absence of documentation on COTS components because the customer will have information about how COTS components are to be used in the host system. To illustrate our approach, we will consider the reliability, maintainability, and availability (RMA) of COTS components as used in host systems.

In addition, COTS suppliers should consider increasing visibility into their products to assist customers in determining the components' fitness for use in a particular application. We offer ideas about information that would be useful to customers and what vendors might do to provide it.

This paper is organized as follows: reliability, maintainability, availability, requirements risk analysis, improved visibility into COTS, cost as the universal COTS metric, and conclusions.

Reliability

There are some intriguing questions concerning how to evaluate the reliability of COTS components that we will attempt to answer [SCH991]. Among these are the following: How do we estimate the reliability of COTS when there is no data available from the vendor? How do we estimate the reliability of COTS when it is embedded in a host system? How do we revise our reliability estimates once COTS has been upgraded? A fundamental problem arises in assessing the reliability of a software component: a software component will exhibit different reliability performance in different applications and environments. A COTS component may have a favorable reliability rating when operated in isolation but a poor one when integrated in a host system. What is needed is the operational profile of COTS components as integrated into the host system in order to provide some clues as to how to test COTS components. We will assume the worst-case situation that documentation and source code are not available. Thus, inspection would not be feasible and we would have to rely exclusively on testing and reliability calculations derived from test data to assess reliability.

The operational profile identifies the criticality of components and their duration and frequency of use. Establishing the operational profile leads to a strategy of what to test, with what intensity, and for what duration. We must recognize that a COTS component must be tested with respect to *both* its operational profile and the operational profile of the host system of which it is a part. The COTS component would be treated like a black box for testing purposes similar to a host component being delivered by design to testing but without the documentation. Testing the COTS components according to these operational profiles will produce failure data that can be used for two purposes: 1) make an empirical reliability assessment of COTS components in the environment of the host system and 2) provide data for estimating the parameters of a reliability model for predicting future reliability [SCH97].

A comprehensive software reliability engineering process is described in [ANS93]. As pointed out by Voas, black box and operational testing alone may be inadequate [VOA98]. In addition, he advocates using fault injection to corrupt one component (e.g., COTS component) to see how well other components (e.g., the host system) can tolerate the failed component. While this approach can identify problems in the software, it cannot fix them without documentation. Thus there must be a contract with the vendor that allows the customer to report problems to the vendor for their resolution. Unfortunately, from the customer's standpoint, vendors are unlikely to agree to such an arrangement unless the customer has significant leverage such as the Federal Government. In the

case where documentation is available, it would be subjected to a formal inspection of its understandability and usability. If the documentation satisfies these criteria, it would be used as an aid to inspecting any source code that might be available. Next we consider COTS maintainability issues.

Maintainability

In the case of maintainability, there are more intriguing issues. Suppose a problem occurs in a host system. Is the problem in COTS or in the host software? Suppose it is caused by an interaction of the two. The customer knows the problem has occurred, but does not know how to fix it if there is no documentation. The vendor, not being on site, does not know the problem has occurred. Even the vendor may not know how to fix the problem if the source of the problem is the host software or an interaction between it and COTS components. In addition, suppose the customer needs to upgrade the host software and this upgrade is incompatible with the COTS components. Or, conversely, the vendor upgrades COTS components and they are no longer compatible with the host software. Lastly, suppose there are no incompatibilities, but the customer may be forced to install the latest COTS components upgrade in order to continue to receive support from the vendor. None of these situations can be resolved without either the customer having documentation to aid in fixing the problem, or a contract with the vendor of the type mentioned above. As in the case of reliability, when neither of these remedies is available, problems can only be identified but they cannot be fixed. Thus the software cannot be maintained. An additional factor that impacts both reliability and maintainability is that the vendor is unlikely to continue to support the software if the customer modifies it. Thus the situation degenerates to one in which the customer is totally dependent on vendor support to achieve reliability and maintainability objectives. This may be satisfactory for office product applications but it is unsatisfactory for mission critical applications. Next we consider the COTS availability issues.

Availability

High availability is crucial to the success of a mission critical system. What will be system availability using COTS? To attempt to answer this question, it is useful to consider hardware as a frame of reference. The ultimate COTS is hardware; it has interchangeable and replacement components. Maintenance costs are kept low and availability is kept high by replacing failed components with identical components. Unlike hardware, availability cannot be kept high by "replacing" the software. A failed component cannot be replaced because the replacement component would have the same fault as the failed component. Fault tolerant software is a possibility but it has had limited success. We see that availability is a function of reliability and maintainability as related by the formula:

Availability = MTTF/(MTTF+MTTR) =

1/1+(MTTR/MTTF),

where MTTF is mean time to failure and MTTR is mean time to repair. MTTF is related to reliability and MTTR is related to maintainability. For high availability, we want to drive *time to failure* to infinity and *repair time* to zero. However, we have seen from the discussion of reliability and maintainability that achieving these objectives is problematic. Thus to achieve high availability, either the COTS software must be of high intrinsic reliability – probably a naive assumption – or there must be in place a strong vendor maintenance program (this assumption may be equally naive). Next we consider COTS visibility issues.

Improved Visibility into COTS

Major drawbacks of including COTS in a software system are the lack of visibility into how the COTS components were developed and an incomplete understanding of the components' behavioral properties [SCH991]. Without this information, it is difficult to assess COTS components to determine their fitness for a particular application. As suggested by McDermid in [TAL98], a partial solution might be for COTS vendors to identify a set of behavioral properties that should be satisfied by the software, and then certifying that those properties are satisfied. For instance, an operating system supplier might certify that a lower-priority task does not interrupt a higher priority task as long as the higher priority task holds the resources required to continue processing. COTS vendors might also include the specifications of those components as well as

details of verification activities in which those specifications had been used to show that specific behavioral properties of the software were satisfied. For instance, an effort in progress at the Jet Propulsion Laboratory [JPL98] involves developing libraries of reusable specifications for spacecraft software components using the PVS specification language [SRI98]. The developers of the libraries work cooperatively with anticipated customers to develop the specifications and identify those properties that the components should satisfy. As they develop the libraries, the component developers use the PVS theorem proverb to show that the behavioral properties are satisfied by the specification. These proofs are intended to be distributed with the libraries. When customers modify the libraries, perhaps to customize them for a new mission, they will be able to use the accompanying proofs as a basis for showing that the modified specification exhibits the desired behavioral properties. Similarly, commercial vendors could work with existing and potential customers through user groups to discover those behavioral properties in which users are the most interested, and then work to certify that their components satisfy those properties. Next we present a methodology for analyzing requirements risk when COTS is embedded in a host system.

Requirements Risk Analysis

In this section we first describe the Shuttle risk management process. Then we consider how it could be modified to accommodate the use of COTS. In providing this analysis, it should not be inferred that we necessarily advocate the use of COTS on the Shuttle or on any other safety critical system. Whether COTS should be employed would depend upon many environmental and application factors. Rather, our goal is to investigate whether the Shuttle risk analysis process is adaptable to the use of COTS.

Shuttle Risk Management Process

One of the software development and maintenance problems of the NASA Space Shuttle Flight Software organization is to evaluate the risk of implementing requirements changes. These changes can affect the reliability, availability and maintainability of the software. To assess the risk of change, a number of risk factors are used. The risk factors were identified by agreement between NASA and the development contractor based on assumptions about the risk involved in making changes to the software. This formal process is called a risk assessment. No requirements change is approved by the change control board without an accompanying risk assessment. During risk assessment, the development contractor will attempt to answer such questions as: "Is this change highly complex relative to other software changes that have been made on the Shuttle?" If this were the case, a high-risk value would be assigned for the complexity criterion. To date this qualitative risk assessment has proven useful for identifying possible risky requirements changes or, conversely, providing assurance that there are no unacceptable risks in making a change.

The following are the definitions of the risk factors, where we have placed the factors into categories and have provided our interpretation of the question the factor is designed to answer. In addition, we added the risk factor *requirements specifications techniques* because we feel that this one could represent the highest reliability risk of all the factors if a technique leads to misunderstanding of the intent of the requirements. For each of the risk factors, we analyze its appropriateness for COTS. As you will see, this analysis not only determines the adaptability of the process to COTS, but also exposes some serious issues in the employment of COTS in *any* system. For example, the Shuttle risk process is all about assessing the risk of requirements changes. In COTS, we would not want to attempt changes because we don't have the necessary source code and other documentation. Furthermore, if we did make a change, it could invalidate our software license. This situation illuminates a serious deficiency in using COTS. Therefore, our only recourse, if feasible, is to change the host software to reflect the change. In other words, COTS has to be used "*as is*" in our system. Thus, in what follows, the risk factors are a function of the change in the host software and how the change relates to and can be integrated with COTS.

In order to modify the Shuttle risk process to make it applicable to the use of COTS, we must change the software change metric from lines of code to components. In addition, we must change our view of the software from a set of individual instructions to a set of interconnected components. Otherwise, it would make no sense

to talk about number of lines of code to be changed in the host software when we only have visibility of COTS at the component level. We will also assume an object oriented development and maintenance paradigm.

Requirements Change Risk Factors

The following are the definitions of the Shuttle risk factors modified to accommodate the use of COTS, where, as mentioned previously, only *host software components* can be changed, but in making the changes, the relationship with COTS components must be considered. If the answer to a yes/no question is "yes", it means this is a high-risk change with respect to the given factor. If the answer to a question that requires an estimate is an anomalous value, it means this is a high-risk change with respect to the given factor. When a change to a component is mentioned below, it will be understood to be a change to host software.

Complexity Factors

- o Qualitative assessment of complexity of change (e.g., very complex)

  - Is this change highly complex relative to other software changes that have been made on the system? What are the interfaces between the host components and COTS components that are affected by the change? Is the change more complex for the host system than for the host software alone?

- o Number of modifications or iterations on the proposed change

  - How many times must the change be modified or presented to the Change Control Board (CCB) before it is approved?

Size Factors

- o Number and types of components affected by the change

  - How many components and types of components must be changed to implement the requirements change?

- o Size of software components that are affected by the change

  - How many component objects are affected by the change?

Criticality of Change Factors

- o Whether the software change is on a nominal or off-nominal component path (i.e., exception condition)

  - Will a change to an off-nominal component path affect the reliability of the software?

- o Operational phases affected by the changed component path (e.g., ascent, orbit, and landing)

  - Will a change to a critical phase of the mission (e.g., ascent and landing) affect the reliability of the software?

Locality of Change Factors

- o The area of the affected change (i.e., critical area such as a component path for a mission abort sequence)

  - Will the change affect objects of components that are critical to mission success?

- o Recent changes to components in the area affected by the requirements change

  - Will successive changes to the components in a given area lead to non-maintainable code?

- o New or existing components that are affected

  - Will a change to new components (i.e., a change on top of a change) lead to non-maintainable software?

- o Number of system or hardware failures that would have to occur before the components that implement the requirement are executed

  - Will the change be on a component path where only a small number of system or hardware failures would have to occur before the changed components are executed ?

Requirements Issues and Function Factors

- o Number and types of other requirements affected by the given requirement change (requirements issues)

  - Are there other requirements that are going to be affected by this change? If so, these requirements will have to be resolved before implementing the given requirement.

- o Possible conflicts among requirements changes (requirements issues)

  - Will this change conflict with other requirements changes (e.g., lead to conflicting operational scenarios)

- o Number of principal software functions and components affected by the change

  - How many major software functions and components will have to be changed to make the given change?

Performance Factors

- o Amount of memory required to implement the change

  - Will the change use memory to the extent that other functions and components will not have sufficient memory to operate effectively?

- o Effect on CPU performance

  - Will the change use CPU cycles to the extent that other functions and components will not have sufficient CPU capacity to operate effectively?

Personnel Resources Factors

- o Number of inspections of components and objects required to approve the change

  - Will the number and duration of inspections be significant?

- o Manpower required to implement the change

  - Will the manpower required to implement the software change be significant?

- o Manpower required to verify and validate the correctness of the change

  - Will the manpower required to verify and validate the software change be significant?

Tools Factor

- o Software tools creation or modification required to implement the change

  - Will the implementation of the change require the development and testing of new tools – for example the development of component and object testing tools?

- o Requirements specifications techniques (e.g., flow diagram, state chart, pseudo code, control diagram).

  - Will the requirements specification method be difficult to understand and translate into components and objects?

As an example, Table 1 shows a partial list of the risk factors compiled for the for the Shuttle *Three Engine Out Auto Contingency* and *Single Global Positioning System* requirements changes.

Table 1

Change Request Number	SLOC Changed	Complexity Rating of Change	Criticality of Change	Number of Principal Functions Affected	Number of Modifications Of Change Request	Number of Requirements Issues	Number of Inspections Required	Manpower Required to Make Change
107734	1933	4	3	27	7	238	12	209.3 MW

Discussion

Although we believe we have made a reasonable translation from a code oriented requirements risk analysis to a component oriented one, it is not clear that the resultant risk model would be entirely usable because no matter how we define the software entities of interest, we still do not have equal visibility of the host

software and COTS. We suggest this is a fundamental problem that has not been solved by COTS advocates, particularly for safety critical systems. Next we present a framework for identifying and analyzing the cost of COTS.

Cost as the Universal COTS Metric

We focus on factors that the user should consider when deciding whether to use COTS software [SCH992]. We take the approach of using the common denominator *cost*. This is done for two reasons: first, cost is obviously of interest in making such decisions and second a single metric – cost in dollars – can be used for evaluating the pros and cons of using COTS. The reason is that various software system attributes, like acquisition cost and availability (i.e., the percentage of scheduled operating time that the system is available for use), are non-commensurate quantities. That is, we cannot relate quantitatively "a low acquisition cost" with "high availability". These units are neither additive nor multiplicative. However, if it were possible to translate availability into either a cost gain or loss for COTS software, we could operate on these metrics mathematically. Naturally, in addition to cost, the user application is key in making the decision. Thus one could develop a matrix where one dimension is *application* and the other dimension is the various *cost elements*. We show how cost elements can be identified and how cost comparisons can be made over the *life* of the software. Obviously, identifying the costs would not be easy. The user would have to do a lot of work to set up the decision matrix but once it was constructed, it would be a significant tool in the evaluation of COTS. Furthermore, even if all the required data cannot be collected, having a framework that defines software system attributes would serve as a user guide for factors to consider when making the decision about whether to use COTS software or in-house developed software. Note that host software could be developed either in-house or under contract. If the former, the in-house cost element below apply to host software.

Certainly, different applications would have varying degrees of relationships with the cost elements. For example, flight control software would have a stronger relationship with the cost of unavailability than a spreadsheet application. Conversely, the latter would have a stronger relationship with the cost of inadequacy of tool features than the former. Due to the difficulty of identifying specific COTS-related costs, our initial approach is to identify cost elements on the ordinal scale. Thus, the first version of the decision matrix would involve ordinal scale metrics (i.e., the cost of unreliability is more important for flight control software than for spreadsheet applications). As the field of COTS analysis matures and as additional data is collected about the cost of using COTS, we will be able to refine our metrics to the ratio scale (e.g., the cost of unreliability in a host system is two times that in a commercial COTS system).

The cost elements for comparing COTS software with in-house software are identified below. This list is not exhaustive; its purpose is to illustrate the approach. These elements apply whether we are comparing a system comprised of all COTS components with all in-house components or comparing only a subset of COTS components with corresponding in-house components. Explanatory comments are made where necessary. Mean values are used for some quantities in the initial framework. This is the case because it will be a challenge to collect *any* data for some applications. Therefore, the initial framework should not be overly complex. Variance and statistical distribution information could be included as enhancements if the initial framework proves successful.

Cost Elements

$C_c(j)$ = Cost of acquiring COTS software in year j.

$C_i(j)$ = Cost of developing in-house software in year j.

$U_c(j)$ = Cost of upgrading COTS software in year j.

$U_i(j)$ = Cost of upgrading in-house software in year j.

$P(j)$ = Cost of personnel who use the software system in year j. This quantity represents the value to the customer of using the software system.

$M_c(j)$ = Cost per unit time of repairing a fault in COTS software in year j. This is the cost of customer time involved in resolving a problem with the vendor.

$M_i(j)$ = Cost per unit time of repairing a fault in in-house software in year j.

$R_c(j)$ = Mean time of repairing a fault that causes a failure in COTS software in year j. This is the average time that the user spends in resolving a problem with the vendor.

$R_i(j)$ = Mean time of repairing a fault that causes a failure in in-house software in year j.

$T(j)$ = Scheduled operating time for the software system in year j.

$A_c(j)$ = Availability of software system that uses COTS software in year j.

$A_i(j)$ = Availability of software system that uses software developed in-house in year j.

These quantities are the fractions of T(j) that the software system is available for use.

$F_c(j)$ = Failure rate of COTS software in year j.

$F_i(j)$ = Failure rate of in-house software in year j.

These quantities are the number of failures per year that cause loss of productivity and availability of the software system.

In some applications, some or all of the above quantities may be known or assumed to be constant over the life of the software system. Using the above cost elements, we derive the equations for the annual costs of the two systems and the difference in these costs. In the cost difference calculations that follow, a positive quantity is favorable to in-house development and a negative quantity is favorable to COTS.

Cost of Acquiring Software

Difference in annual cost = $C_c(j) - C_i(j)$ (1)

Cost of Upgrading Software

Difference in annual cost = $U_c(j) - U_i(j)$ (2)

Cost of Software being Unavailable for Use

Annual cost of COTS software being unavailable for use = $(1-A_c(j)) * P(j)$.

Annual cost of the in-house software being unavailable for use = $(1-A_i(j)) * P(j)$.

Difference in annual cost =
$P(j) * (A_i(j) - A_c(j))$ (3)

Cost of Repairing Software

Average annual cost of repairing failed COTS software = $F_c(j) * T(j) * R_c(j) * M_c(j)$.

Average annual cost of repairing failed in-house software = $F_i(j) * T(j) * R_i(j) * M_i(j)$.

Difference in annual cost =

$T(j) * ((F_c(j) * R_c(j) * M_c(j)) - ((F_i(j) * R_i(j) * M_i(j))$ (4)

Then, TC_j, total difference in cost in year j, is the sum of (1), (2), (3), and (4). Because there is the opportunity to invest funds in alternate projects, costs in different years are not equivalent (i.e., funds available today have more value than an equal amount in the future because they could be invested today and earn a future return). Therefore, a stream of costs over the life of the software for n years must be discounted by k, the rate of return on alternate use of funds. Thus the total discounted cost differential between COTS software and in-house software is:

$$\sum_1^n TC_j/(1+k)^j$$

In this initial formulation, we have not included possible differences in functionality between the two approaches. However, a reasonable assumption is that COTS software would not be considered unless it could provide minimum functionality to satisfy user requirements. Thus, a typical decision for the user is whether it is worth the additional life cycle costs to develop an in-house software system with all the desirable attributes.

Conclusions

The decision to employ COTS on mission critical systems should not be based on development cost alone. Rather, costs should be evaluated on a total life cycle basis and RMA should be evaluated in a system context (i.e.,

COTS components embedded in a host system). COTS suppliers should also consider making available more detailed information regarding the behavior of their systems, and certifying that their components satisfy a specified set of behavioral properties. In addition, a formal risk assessment of requirements should be performed taking into account the characteristics of host system environments.

References

[ANS93] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[CLE97] Clemins, Archie, "IT-21: The Path to Information Superiority." CHIPS Jul 1997, http://www.chips.navy.mil/chips/archives/97_jul/file.htm, p. 1.

[JPL98] "Reusable Libraries of Formal Specifications", NASA Formal Methods web site, http://eis.jpl.nasa.gov/quality/Formal_Methods/library.html, 1998.

[KOH99] Ronald J. Kohl, "V&V of COTS Dormant Code: Challenges and Issues", Proceedings of the First Workshop on Ensuring Successful COTS Development, 21st International Conference on Software Engineering, Los Angeles, California, May 22nd, 1999, 2 pages.

[SCH97] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp.88-98.

[SCH991] Norman F. Schneidewind and Allen P. Nikora, "Issues and Methods for Assessing COTS Reliability, Maintainability, and Availability", Proceedings of the First Workshop on Ensuring Successful COTS Development, 21st International Conference on Software Engineering, Los Angeles, California, May 22nd, 1999, 4 pages.

[SCH992] Norman F. Schneidewind, "Cost Framework for COTS Evaluation", Proceedings of COMPSAC 99, Phoenix, AZ, 27 October 1999, pp. 100-101.

[SRI98] "The PVS Specification and Verification System", SRI International Computer Science Laboratory, http://www.csl.sri.com/sri-csl-pvs.html, 1998.

[TAL98] Nancy Talbert, "The Cost of COTS", IEEE Computer, Vol. 31, No. 6, June 1998, pp. 46-52.

[VOA98] Jeffrey M. Voas, "Certifying Off-the-Shelf Software Components", IEEE Computer, Vol. 31, No. 6, June 1998, pp. 53-59.